$x \sim p(x) = N(0, R)$ (covariance matrix)

$R = E[xx^t] = E\left[\square\right] \rightarrow R_{ij} = E[x_i x_j]$ (outer product; scalar)

R is square & invertible

$a^t R a = a^t E[xx^t] a = E[(a^t x)(x^t a)] = E[z^2] > 0$

↳ R is positive definite

$R \cong \hat{R} = \frac{1}{N} \sum_{i=1}^{N} y_i y_i^t$

Multivariate Gaussian Distribution

Used a lot, in M.L. and elsewhere

- Let $\mathbf{x}$ be a zero-mean random variable on $\mathbb{R}^p$

General Form

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}} |R|^{-1/2} \exp\left\{ -\frac{1}{2} \mathbf{x}^T R^{-1} \mathbf{x} \right\}$$

($\vec{x} \in \mathbb{R}^p$; vector; quadratic form)

as exponent gets larger, $p(\vec{x})$ gets smaller

x^T R^{-1} x

if not zero-mean: $\dots \exp\left\{ -\frac{1}{2} (\vec{x} - \vec{\mu})^T R^{-1} (\vec{x} - \vec{\mu}) \right\}$

where R is the $p \times p$ covariance matrix.

- The matrix R is a positive definite symmetric matrix, then

$$R = E \Lambda E^t$$

↳ because $R_{ij} = E[x_i x_j] = E[x_j x_i] = R_{ji}$

where $E = [\mathbf{e}_1, \ldots, \mathbf{e}_p]$ is an orthonormal matrix of eigenvectors, and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_p)$ is a diagonal matrix of eigenvalues. (λ's positive)

$Re_i = E \Lambda E^t e_i = E \Lambda \begin{bmatrix} e_1^t \\ \vdots \\ e_p^t \end{bmatrix} e_i$

$= E \Lambda \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = E \begin{bmatrix} 0 \\ \vdots \\ \lambda_i \\ \vdots \\ 0 \end{bmatrix} = \lambda_i e_i$ ✡

- Therefore, we have $E^t E = I$.

$E = [e_1 \ \ldots \ e_p]$

$e_i^t e_j = \delta(i-j) \rightarrow$ orthonormal matrix

$R > 0$ i.i.f. $\lambda_i > 0 \ \forall \ i$

$e^{j\omega n} \rightarrow$ LTI $\rightarrow \lambda_\omega e^{j\omega n}$

$H(\omega) = \lambda_\omega$

sine waves are eigenfunctions for LTI systems (eigenvectors)

Re-Watch:

- integral of $p(\vec{x})$ has to be 1, that's why there is a coefficient in front of the exp{}

- R is symmetric & positive definite
  $R = R^+$, $\forall x \in \mathbb{R}^p$, $x^+ R x > 0$

$$E^+ E = I: \quad \begin{bmatrix} e_1^+ \\ \vdots \\ e_p^+ \end{bmatrix} [e_1 \dots e_p] = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$$

$$e_i^+ e_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} = \delta(i-j)$$

$R = E \Lambda E^+$ - works because R is symmetric
- E & Λ are real-valued

Idea: $Re_i = \lambda_i e_i$

E - eigentransform
Λ - eigenvalues, all elements are greater than zero (R positive definite)

$R^{-1} = (E \Lambda E^t)^{-1}$ can invert R just by inverting the eigenvalues
$= E \Lambda^{-1} E^t$

Contour for the Energy Function

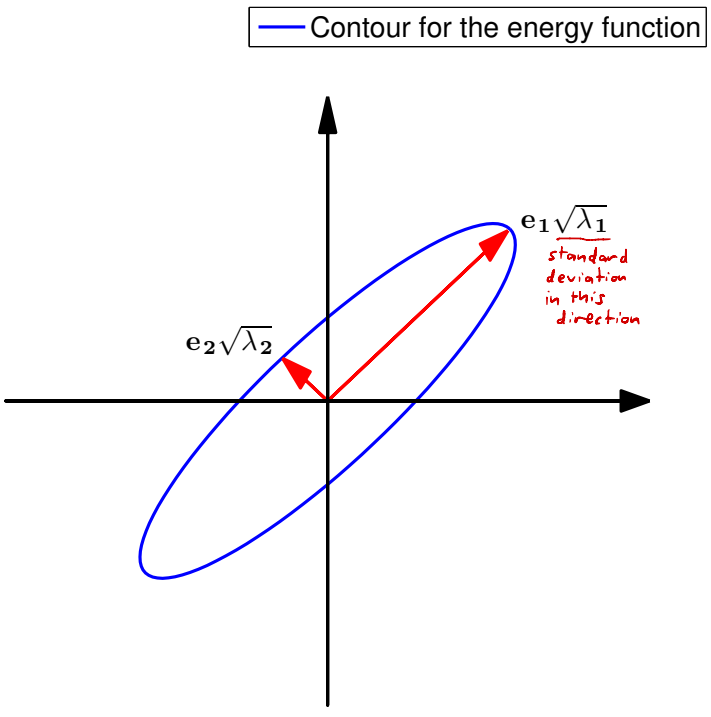
- Intuitively, the energy function (square of the Mahalanobis distance) quadratic form

$$f(\mathbf{x}) = \mathbf{x}^t R^{-1} \mathbf{x}$$

f(0) = 0 (minimum)

has contour plots shown here.

let $\lambda_1 \geq \lambda_2$

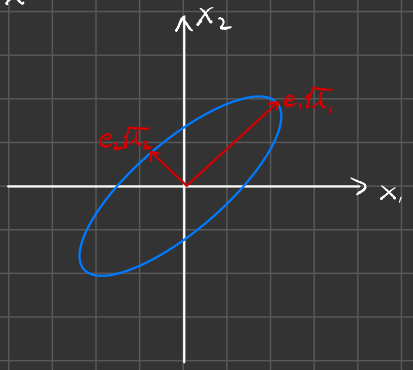


Scalar Case: $p(x) = \frac{1}{2} \exp\left\{-\frac{1}{2\sigma^2} x^2\right\}$

if eigenvalues negative, standard deviation is negative, which doesn't make sense

$f(\vec{x}) = \vec{x}^+ R^{-1} \vec{x}$

$f(\vec{0}) = \vec{0}$



$f(\vec{x}) = -\log(p(\vec{x})) + c$

as $p(\vec{x})$ goes down, $f(\vec{x})$ increases

$\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_p)$

$\lambda_i \geq \lambda_{i+1}$

- eigenvectors e_i are principal axes
- eigenvalues specify the length to travel along the principal axes

- if R diagonal, components are uncorrelated, and each diagonal element is the variance of a single component

Let $X \sim N(0, R)$

$E[\vec{x}\vec{x}^+] = R$ - by definition

$$\hat{R} = \frac{1}{M} \sum_{k=1}^{M} x_k x_k^+$$

($\hat{R}$: estimate (r.v.))

Law of large numbers $\rightarrow \lim_{M \to \infty} \hat{R} \overset{a.s.}{=} R$

$R = E \Lambda E^+$

(E: eigenvectors, Λ: eigenvalues)

E is orthonormal: $EE^+ = E^+E = I$

Gaussian distributions with zero mean have symmetry

$E = [e_1 \ldots e_p]$

E can be viewed as a rigid body rotation

Gaussian Random Variable Decorrelation

$\vec{x} \sim N(0, R)$; $R = E \Lambda E^t$

- Consider $\tilde{\mathbf{x}} = E^t \mathbf{x}$, then

transformed version of x

rigid rotation

$$\begin{aligned}
\mathbb{E}[\tilde{\mathbf{x}}\,\tilde{\mathbf{x}}^t] &= \mathbb{E}[E^t \mathbf{x}\,\mathbf{x}^t E] \\
&= E^t \mathbb{E}[\mathbf{x}\,\mathbf{x}^t] E \\
&= E^t R E \\
&= E^t E \Lambda E^t E \\
&= \Lambda
\end{aligned}$$

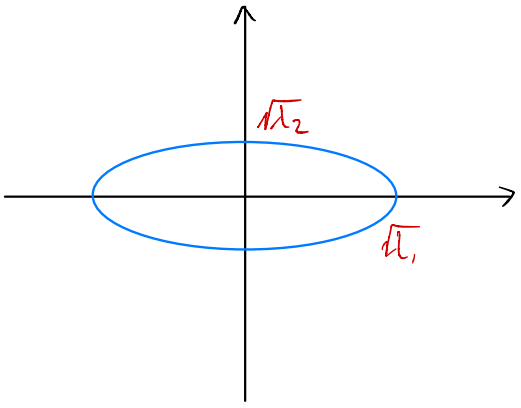
bring out E's as they are deterministic

- Therefore, the elements of $\tilde{\mathbf{x}}$ are uncorrelated with variance $\mathbb{E}[\tilde{x}_i^2] = \Lambda_{ii}$. , zero elsewhere → $\tilde{x}$ elements uncorrelated but not all same variance

Therefore,

- The elements of $\tilde{\mathbf{x}}$ are independent, since $\tilde{\mathbf{x}}$, as the linear transform of $\mathbf{x}$, is Gaussian distributed.

because x is Gaussian & zero-mean



$R = E[xx^+]$

$= E \Lambda E^+$

$R \cong \hat{R} = \frac{1}{N} \sum_{i=1}^{N} y_i y_i^+$

sample covariance

$X = [y_1 \cdots y_N]$ sample vectors

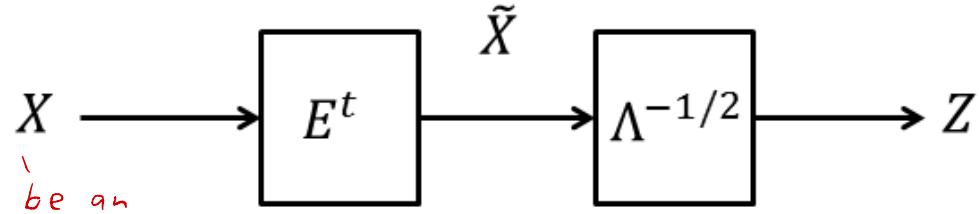
$10^6 \times 10^2$

$p \times N$

$\hat{R} = \frac{1}{N} X X^+ = \frac{1}{N} \sum_{i=1}^{N} y_i y_i^+$

Whitening Gaussian Random Variables

"Sphering"



can be an image

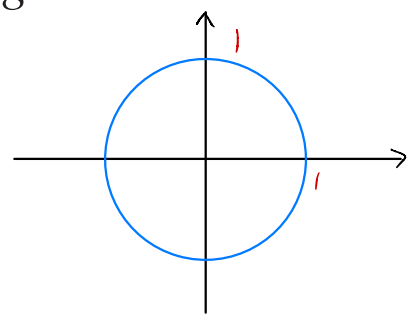
$$\mathbb{E}[Z \; Z] = I$$

multiplies on the left

- So E^t decorrelates $\mathbf{x}$, while $\Lambda^{-\frac{1}{2}} E^t$ whitens $\mathbf{x}$.

$$E^t \mathbf{x} = \begin{bmatrix} \mathbf{e}_1^t \\ \vdots \\ \mathbf{e}_p^t \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

- The eigenvectors $\mathbf{e}_k$, called eigen-signals, are basis vectors to represent the signal $\mathbf{x}$.
- If $\mathbf{x}$ represents an image, then the eigenvectors $\mathbf{e}_k$ are also called *eigenimages*.



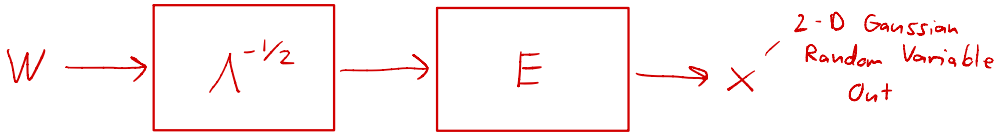
$X = U \Sigma V^+$ (works for non-symmetric matrices)

U: e-vects of R; V: e-vals of R

Turns out that

# of pixels; each training image

X^+



Arbitrary Generator

Eigenimage Estimation

- Assume we have n training vectors $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, then

$$\begin{aligned} R_x =& E[\mathbf{x}_k\, \mathbf{x}_k^t] \\ =& E[XX^t]/n \\ \cong& XX^t/n \\ =& S \end{aligned}$$

  where $S = \frac{1}{n} XX^t$ is the sample correlation matrix

$$S_{ij} = \frac{1}{n} \sum_{k=1}^{n} X_{ik} X_{jk}$$

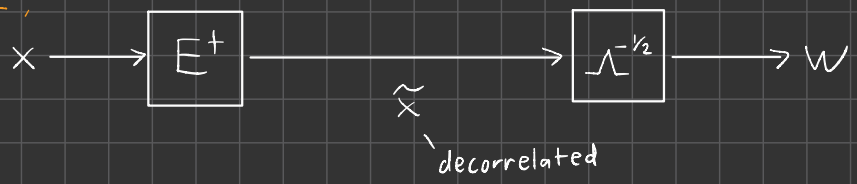
- Decompose S as

$$S = \hat{E} \hat{\Lambda} \hat{E}^t$$

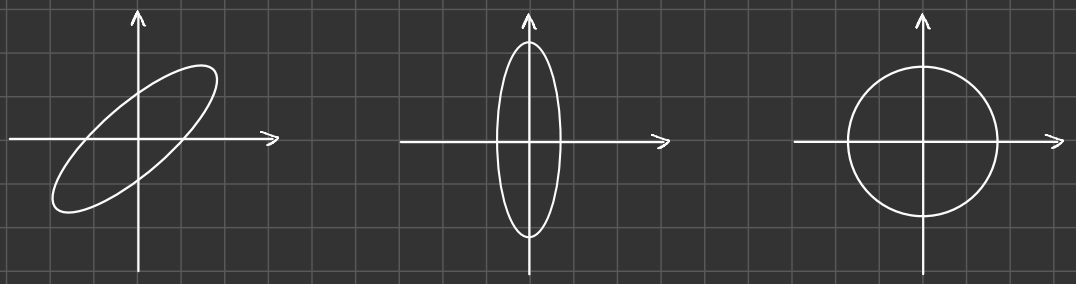
  where $\hat{E}$ is an estimate of the eigenvectors, and $\hat{\Lambda}$ is an estimate of the eigenvalues.

- E could be very large, especially when X represents n images.

Analysis:



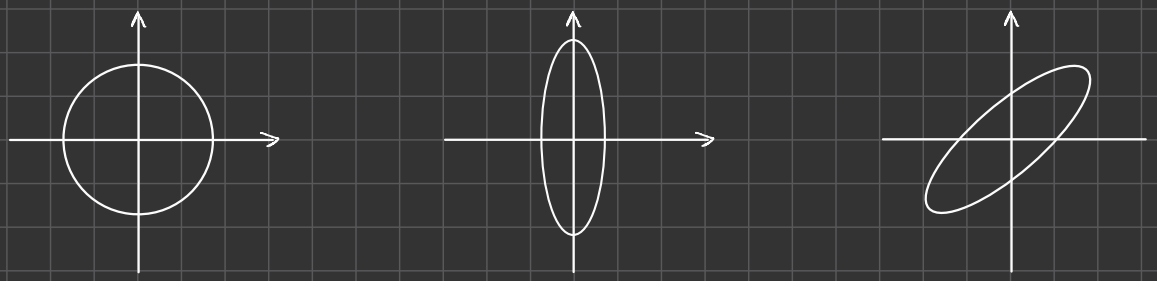
Hypothesis Testing



correlated | uncorrelated | white, i.i.d noise (mean: 0, variance: 1)

Synthesis:

Arbitrary Generator



$X \sim N(0, R)$
$R = E\Lambda E^+$
$W = \Lambda^{-1/2} E^+ X$; $X = E\Lambda^{1/2} W$

$W \sim N(0, I)$
$R = I$, which is diagonal → data points uncorrelated
↳ uncorrelated + Gaussian: data points independent

W is a column vector of length p, just like X
- generating W on a computer is straightforward

- components of $\tilde{X}$ are 0-mean and independent, but they don't have the same variance/standard deviation

Ex: Generate $X \sim N(\mu, R)$

Step 1) $R = E \Lambda E^t$ - call routine to calculate E, Λ from R

Step 2) $W \sim (0, I)$ - call rand function p times to generate W

Step 3) $\tilde{x} = \Lambda^{1/2} W$ $\Lambda^{1/2}$ are the standard deviations?

Step 4) $x' = \tilde{y} = \sum_{i=1}^{p} e_i \tilde{x}_i$

Step 5) $X = X' + \mu$

For large $X \in \mathbb{R}$

Singular Value Decomposition (SVD)

not the same as eigendecomposition

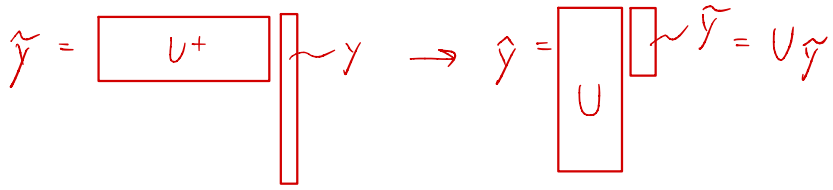
- For $n < p$ it looks like

eigenimages are the columns

$$\underbrace{\begin{bmatrix} X \\ \mathbb{X} \\ p \times n \end{bmatrix}}_{\text{raster-order training images}} = \begin{bmatrix} U \\ \\ p \times n \end{bmatrix} \begin{bmatrix} \Sigma \\ n \times n \end{bmatrix} \begin{bmatrix} V^t \\ n \times n \end{bmatrix}$$

  - The columns of U are orthonormal and called left hand singular vectors. $\rightarrow U^t U = \mathbb{I}_n$
  - The columns of V are orthonormal and called right hand singular vectors. $\rightarrow V^t V = \mathbb{I}_n$
  - Σ is diagonal matrix of singular values.

Any Linear Transformation
1) rigid body transformation #1
2) squeezing & stretching
3) rigid body transformation #2



$\tilde{w} = T \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

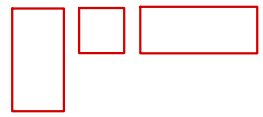
$E[\tilde{y}\tilde{y}^+] = R_y = \Sigma^2$ 'like the Fourier coefficients

$E[\tilde{w}\tilde{w}^+] = E[Tww^+T^+] = TE[ww^+]T^+ = TIT^+ = I$

Eigenimage Estimation using SVD

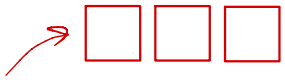
- Notice that

$$\begin{aligned} XX^t &= U\Sigma V^t V \Sigma U^t \\ &= U\Sigma^2 U^t \end{aligned}$$



So U is the set of the desired eigenvectors of XX^t.

- How to compute U?

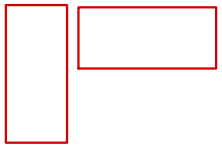


  - Notice that $X^tX = V\Sigma^2V^t$ is a $n \times n$ matrix, and V contains eigenvectors of a much smaller matrix.
  - Algorithm

V decorrelates the columns of X (the training images)

    * Find eigenvectors V of the matrix X^tX
    * Compute $XV = U\Sigma$, Then the columns of U are the normalized columns of XV, and Σ are the normalization factors.

basis with decreasing weight of eigenvectors

*) Form $B \leftarrow \frac{1}{N} X^+X =$ 

* $(V, \Sigma^2) \leftarrow \text{eigen}(B)$
* $\Sigma = \sqrt{\Sigma^2}$
* $U \leftarrow XV\Sigma^{-1}$ (decorrelates, normalizes)
* eigenvectors $\leftarrow U$, eigenvalues $\leftarrow \Sigma^2$ } represent the image

Common: Let $x_1, \ldots, x_n$ be i.i.d. R.V.

$X = [x_1 \ldots x_n]$ Experimentation: x_j could be an observation

x_i are random vectors $\sim N(0, R)$ ← distribution that you sample from

$$E[x_i x_i^T] = E\left[\square\right] = R = E\left[\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T\right]$$

Expectation is linear, R summed n times then scaled by $1/n$ is simply R

$$\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T = \frac{1}{n} XX^T = \hat{R}$$ ← estimate of R (random)

often: $n < p$ → $X =$  ~ tall & thin

$R = E[\hat{R}]$

↳ $\hat{R}$ is an unbiased estimate

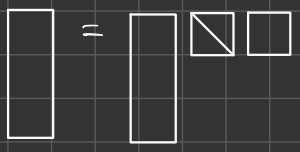
$\hat{R} = \frac{1}{n} XX^T =$ 

if $p = 10^6$, then $\hat{R} \in \mathbb{R}^{p \times p}$

$\hat{R}$ has 10^{12} entries, not very practical

$\hat{R} = E \Lambda E^T$?? → enter SVD

$X = U \Sigma V^T$

 – can always do this decomposition

$\hat{R} = \frac{1}{n} XX^T = U \Sigma^2 U^T$ → the eigendecomposition of $\hat{R}$

$X^T X = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T$

↑ smaller than XX^T

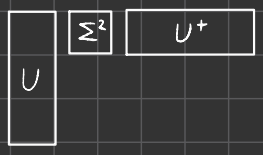
Interpretation: $X^T X$ is taking the correlation of the images, and it is symmetric

Overall Process Steps:

Step 1) Do SVD of X
$X = U\Sigma V^t$, can call in numpy

~ can effectively be considered training data

Step 2) Eigendecomposition $\hat{R} = U\Sigma^2 U^t$



column space of U does not span $\mathbb{R}^n$

Step 3) Use them!

columns of U are eigenvectors (eigenimages in this case)

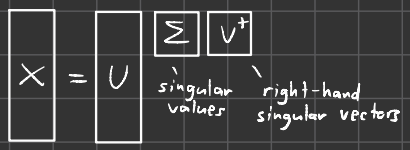
$\tilde{X} = U^t X$

↑ low-dimensional representation of image X

↑ image

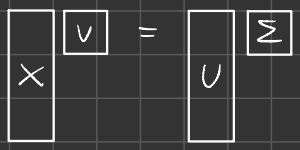
} dimensionality reduction technique ✱

Session 24 continuation from Session 23

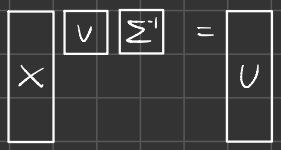


columns of X are the sample vectors

correlated ≡ not orthogonal



V de-correlates the columns of X, resulting columns will be orthogonal but won't have the same norm (some columns will have more energy than others)



Multiply by diagonal matrix → scale each column ✱
If multiplication on LHS → scale each row (as FYI)
Σ^{-1} scales the decorrelated columns of X so they all have the same unit norm

Intuitively, this is sort of like Graham Schmidt

Remember: XX^t / X^tX is exactly the correlation between the rows / columns of X, respectively

$X = U\Sigma V^T$

$\hookrightarrow X^T X = V\Sigma^2 V^T = R$ ← "small" matrix, $n \times n$

this is how V can be calculated

then, decorrelate the columns of X ✡
then, scale with Σ^{-1} to get U

This is how U is calculated ✡

- columns of U are the eigenimages
- if singular values are ordered from largest to smallest, which is pretty conventional, you'll get the eigenimages in order from lowest frequency to highest frequency - why is this ??

Algorithm:

Step 0) Form matrix of training data, used to build model

$X = [x_1 \dots x_n]$ → could be images, signals, etc.

Step 1) Compute U

a) $R \leftarrow X^T X$
b) $R = V\Sigma^2 V^T$
c) $U = XV\Sigma^{-1}$

Step 2) Use the eigenimages!!

$U = [u_1 \dots u_n]$

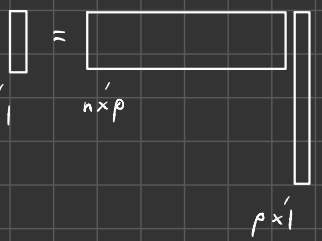
Take new image y

$f_i = u_i^T y = \langle u_i, y \rangle$

f_i: scalar feature

$f = U^T y$

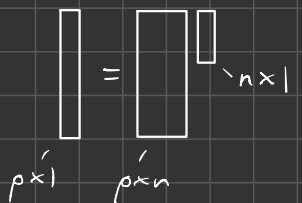
f: feature vector



intuition: f is an n-dimensional representation of a p-dimensional image

dimensionality reduction process

$\tilde{y} = Uf = y + \varepsilon$ (transforming back; ε: residual error, generally nonzero)



f is a set of scalar weights that multiply each column of U, which are the basis vectors, and the result is a representation of y with some residual error

Intuitively sort of like a Fourier Transform, or an image compression algori